Swami Sahajanand College of Computer Science

B.C.A. SEM-V[NEP]

Subject: DATABASE TECHNOLOGY IN INDIA
Major12 - 26516

UNIT-3

ADVANCE SQL*PLUS

1) Data Constrains
2) Types of Data Constrains.
3) In Built Functions: Aggregate, Numeric, String,
4) Data/Time, Conversion.
5) Grouping of Data
6) Sub queries and Types of Sub queries
7) Join and its types
8) Set Operations (Union, Intersect and minus)
9) Schema and Schema objects: View, Sequence, index, synonyms.

Q-1 What is Data Constraints? Also explain types of Data Constraints.

⌗ Data Constraints

- ⌗ Oracle provides a special feature called data constraint/integrity.
- ⌗ Constraint that is applied at the time of creation of data structure.
- ⌗ Only the data which satisfies the constraints rules will be stored in database.
- ⌗ If it is violating the data constraints, it must be rejected.
- ⌗ This ensures that data stored within data structure are valid data.
- ⌗ Constraints could be **column level** or **table level.**
- ⌗ **Column level** constraints are applied only to one column per table.
- ⌗ **Table level** constraints are applied to the whole table.

⌗ Following are commonly used constraints available in SQL.

No.	Constraint Name	Description
1	NOT NULL	Ensures that a column cannot have NULL value.
2	DEFAULT	Provide a default value for a column.
3	UNIQUE	Ensures that all values in a column are different.
4	PRIMARY KEY	Uniquely identified each rows/records in a database table.
5	FOREIGN KEY	Uniquely identified a rows/records in any another database table.
6	CHECK	It ensures that all values in a column satisfy certain conditions.

(1) NULL or NOT NULL Constraint:

⌗ NULL keyword indicates that a column can contain NULL values.
⌗ The NOT NULL constraint specifies that a column cannot contain NULL value.
⌗ To satisfy this constraint NULL, the column can contain NULLs by default.
⌗ The NOT NULL constraint requires that the columns of the table always contain a value.
⌗ Setting a NULL value is appropriate when the actual value is unknown.
⌗ A NULL value is not same as a value zero.
⌗ NULL value can be inserted into columns of any data type.

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size) NULL,
<Column_Name 2><data type> (size) NOT NULL,
 ……..
……..,
<Column_Name N><data type> (size)
);
```

:: Example ::

```
create table student
(
      IDnumber(5),
      NAME varchar2(30) NOT NULL,
      DOB date,
      MOBILE_NO number(10) NULL
);
```

Output: Table created.

⌗ Here, NAME field cannot contain NULL value. Because we used NOT NULL constraint.
⌗ MOBILE_NO can contain NULL value. Because we used NULL constraint.

(2) DEFAULT Constraint:

- The DEFAULT constraint provides a default value to a column.
- At the time of table creation a default value can be assign to it.
- When the user is loading a record with values and leaves this cell empty, the DBA will automatically load this cell with the default value specified.
- The data type of the default value should match the data type of the column.
- If INSERT INTO statement does not provide a specific value, then we used DEFAULT constraint.
- These constraint apply only column level.

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size) default <value>,
<Column_Name 2><data type> (size),
 ……..
……..,
<Column_Name N><data type> (size)
);
```

:: Example ::

```
create table student
(
      IDnumber(5) default 111,
      NAME varchar2(30),
      DOB date,
      MOBILE_NO number(10)
);
```

Output: Table created.

- Here, ID field contain default value like 111.

(3) UNIQUE Constraint:

- The UNIQUE column constraint permits multiple entries of NULL into the column.
- Unique key will not allow duplicate value.
- Unique index is created automatically.
- A table can have more than one unique key which is not possible in PRIMARY KEY.
- Unique key can combine upto 16 columns in a Composite Unique Key.
- Unique key cannot be possible in LONG or LONG RAW data type.

- **Column Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size) UNIQUE,
<Column_Name 2><data type> (size),
 ……..
<Column_Name N><data type> (size)
);
```

:: Example ::

```
create table student
(
      IDnumber(5) UNIQUE,
      NAME varchar2(30),
      DOB date,
      MOBILE_NO number(10)
);
```

- **Table Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size),
<Column_Name 2><data type> (size),
 ……..
<Column_Name N><data type> (size),
UNIQUE (< Column_Name 1>,<Column_Name 2>)
);
```

:: Example ::

```
create table student
(
      ID number(5) UNIQUE,
      NAME varchar2(30),
      DOB date,
      MOBILE_NO number(10),
      UNIQUE (ID, NAME)
);
```

(4) PRIMARY KEY Constraint:

"A PRIMARY KEY is used to uniquely identify each row in a table."

- A primary key is one or more column in a table.
- A primary key values must not be NULL and must be unique across the column.
- When you define any column as primary key it becomes a mandatory column.
- The column cannot be left blank.
- If single column is not sufficient to uniquely identify the row, you can use combination of two or more columns to uniquely identify a row.
- This combination of primary key is known as composite primary key.
- A table can have only one primary key.

PRIMARY KEY= UNIQUE+ NOT NULL

- **Features of Primary key:**

- Primary key is a column or a set of columns that uniquely identifies a row.
- It main purpose is the record uniqueness.
- Primary key will not allow duplicate values.
- Primary key will also not allow NULL values.
- Primary key is not compulsory but it is recommended.
- Primary key helps to identify one record from another record also helps in relation of table.
- Primary key cannot be possible in LONG or LONG RAW datatype.
- Only one primary key is allowed per table.
- Unique index is created automatically if there is a primary key.
- One table can combine upto 16 columns in a composite primary key.

- **Column Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size) PRIMARY KEY,
<Column_Name 2><data type> (size),
 ……..
<Column_Name N><data type> (size)
);
```

:: Example ::

```
create table student
(
      IDnumber(5) PRIMARY KEY,
      NAME varchar2(30),
      DOB date,
      MOBILE_NO number(10)
);
```

- Here, only one filed ID has a primary key.

- **Table Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size),
<Column_Name 2><data type> (size),
 ……..
<Column_Name N><data type> (size),
PRIMARY KEY (< Column_Name 1>,<Column_Name 2>)
);
```

:: Example ::

```
create table student
(
      ID number(5) UNIQUE,
      NAME varchar2(30),
      DOB date,
      MOBILE_NO number(10),
      PRIMARY KEY (ID, NAME)
);
```

- Here, ID and NAME both field have primary key.

(5) FOREIGN KEY Constraint:

- Foreign keys represent relationship between tables.
- A foreign key is a column whose values are derived from the primary key of some other table.
- The table in which the foreign key is defined is called a **foreign table** or **detail table**.
- The table that define the primary key or unique key and is referenced by the foreign key is called the **primary table** or **master table**.
- A foreign key can be defined in either a create table statement or an alter table statement.
- REFERENCES keyword is used for referencing table.

- **Features of Foreign key:**

- Data type for relevant column in master and detail table must be same.
- Parent that is being referenced has to be unique or primary key.
- Child may have duplicates and nulls but unless it is specified.
- Foreign key constraint can be specified on child but not on parent.
- Deleting record from master table is not allowed if corresponding records are available in detail table.
- If ON DELETE CASCADE option is set delete operations on master will delete all records from detail table.
- Relationship can be established with primary key or unique key columns in master table.

- **Column Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size) REFERENCES <Table_Name> (Column_Name),
<Column_Name 2><data type> (size),….., <Column_Name N><data type> (size)
);
```

:: Example ::

```
create table marksheet
(
      ID number(5) REFERENCES student(ID), Total number(3), Percentage number(5,2),
);
```

- **Table Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size), <Column_Name 2><data type> (size),……,
<Column_Name N><data type> (size),
FOREIGN KEY (Column1) REFERENCES <Table_Name> (Column_Name),
FOREIGN KEY (Column2) REFERENCES <Table_Name> (Column_Name)
);
```

:: Example ::

```
create table marksheet
(
        ID number(5), EID number(5), Total number(3), Percentage number(5,2),
        FOREIGN KEY (ID) REFERENCES student(ID),
        FOREIGN KEY (EID) REFERENCES employee(EID)
);
```

(6) CHECK Constraint:

- Business rule validation can be applied to a table column by using CHECK constraints.
- If you want to keep values of a column within a certain range then CHECK constraint is used.
- CHECK constraints must be specified as a logical expression that evaluate either True or False.
- For example: In a bank table balance column contains value >=500.
- So the condition defined in the constraint and permits the INSERT or UPDATE of the row in the table if the condition is satisfied

- **Column Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size) CHECK (Logical Expression),
<Column_Name 2><data type> (size),....., <Column_Name N><data type> (size)
);
```

:: Example ::

```
create table marksheet
(
        ID number(5) CHECK(ID>0), Total number(3), Percentage number(5,2),
);
```

- Here, ID field contain value greater than zero.

- **Table Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size), <Column_Name 2><data type> (size),……..,
<Column_Name N><data type> (size),
CHECK (Column1 [Logical Expression]), CHECK (Column2 [Logical Expression]),……..
);
```

:: Example ::

```
create table student
(
      ID number(5), NAME varchar2(30), EID number(3), BALANCEnumber(10,2),
      CHECK(ID>0), CHECK(NAME=UPPER(NAME)), CHECK(BALANCE>=500)
);
```

- Here, ID field contain value greater than zero.
- NAME must be entered in Capital Letters only.
- BALANCE field contain value >=500.

Q-2 Explain InBuilt Functions: Aggregate, Numeric, String, Date/Time, Conversion in detail.

❖ **DUAL:**

- Dual is a small oracle inbuilt table. The table is owned by SYS and it is available for the all users.
- The dual table provides only one row and one column in it.
- It supports arithmetic calculation and data formatting.
- It is also known as DUMMY table.
- **Example**: - select 4*5 “multiply” from dual;
- **Output**: - multiply 20
- **Example**: - select sysdate from dual; **Output**: - 23-Jun-2018

❖ **FUNCTION:**

- Function are pre-defined set of subroutines that may operate on one or more rows.
- Basically a function takes some data input as an argument.
- Processes that data and returns some values as a result.
- Function can divided into following categories:
    1) Aggregate Functions.
    2) Numeric Functions.
    3) String Functions.
    4) Date/Time Functions.
    5) Conversion Functions.

(1) Aggregate Functions:

1) sum():

Syntax	sum(value)
Purpose	It returns the total of given numeric expression.
Example	select sum(salary) "Total_salary" from emp;
Output	Total_salary 29000

2) avg():

Syntax	avg(value)
Purpose	It returns the average value of N and it ignore the NULL value.
Example	select avg(salary) from emp;
Output	5000.589

3) min():

Syntax	min(value)
Purpose	It returns the minimum values from given list of values.
Example	select min(salary) from emp;
Output	500

4) max():

Syntax	max(value)
Purpose	It returns the maximum values from given list of values.
Example	select max(salary) from emp;
Output	50000

5) count(*):

Syntax	count(*)
Purpose	It returns total number of records in the table, including duplicate and null value.
Example	select count(*) "total no of rows" from emp;
Output	total no of rows 10

6) count():

Syntax	count(value)
Purpose	It returns total number of records based on value, if value is null it will not be included in counting.
Example	select count(salary) "total no of salary" from emp;
Output	total no of salary 8

(2) Numeric Functions:

1)abs():

Syntax	abs(value)
Purpose	Returns absolute value of the given number
Example	select abs(-15)from dual;
Output	15

2) ceil():

Syntax	ceil(value)
Purpose	Returns smallest integer value that is greater than or equal to given value.
Example	select ceil(15.20)from dual;
Output	16

3)floor():

Syntax	floor(value)
Purpose	Returns smallest integer value that is less than or equal to given value.
Example	select floor(15.20)from dual;
Output	15

4)mod():

Syntax	mod(value, divisor)
Purpose	It returns remainder value.
Example	select mod(5,2)from dual;
Output	1

5) power():

Syntax	power(x, n)
Purpose	It returns x raise to n value. (x^n)
Example	Select power(5,2)from dual;
Output	25

6)sqrt():

Syntax	sqrt(value)
Purpose	It returns square root of given value.
Example	select sqrt(25) from dual;
Output	5

7)round():

Syntax	round(value, precision)
Purpose	It rounds the value up to given precision value.
Example	select round(15.456,2)from dual;
Output	15.46

8) trunc():

Syntax	trunc(value, precision)
Purpose	It returns a number truncated to a certain number of decimal places.
Example	select trunc(15.456,1)from dual;
Output	15.4

9)sign():

Syntax	sign(value)
Purpose	This function tells you the sign of value. 1 for positive value. -1 for negative value and 0 for 0 value.
Example	select sign(-5)from dual;
Output	-1 [sign(-5)=-1, sign(5)=1, sign(0)=0]

10) greatest():

Syntax	greatest (expr1,expr2, …)
Purpose	Returns the greatest [Highest] value from list of expression.
Example	select greatest(10,30,20) from dual;
Output	30

11) least():

Syntax	least (expr1,expr2, …)
Purpose	Returns the least [Lowest] value from list of expression.
Example	select least(10,30,20) from dual;
Output	10

12) exp():

Syntax	exp(value)
Purpose	Returns e raised to the n^{th} power, where e=2.71828183.
Example	select exp(5)from dual;
Output	148.413159 [2.7182^5]

(3) String Functions:

1) upper():

Syntax	upper(string)
Purpose	It returns set of character with all the letters in the upper case.
Example	select upper('oracle') from dual;
Output	ORACLE

2) lower():

Syntax	lower(string)
Purpose	It returns set of character with all the letters in the lower case.
Example	select upper('ORACLE') from dual;
Output	Oracle

3) initcap():

Syntax	initcap(string)
Purpose	Returns a string with the first letter of each word in upper case.
Example	select initcap('hello how are you') from dual;
Output	Hello How Are You

4) substr():

Syntax	substr(string, m,n)
Purpose	It returns the specific part of a given string. M indicates the starting position and N indicates total number of character.
Example	Select substr('ORACLE',3,4) from dual;
Output	ACLE

5) length():

Syntax	length(string)
Purpose	It returns the number of character in a given string.
Example	Select length('BCA') from dual;
Output	3

6) ltrim():

Syntax	ltrim(string, [character_set])
Purpose	Removes characters from the left of char with initial characters removed upto the first character not in set.
Example	Select ltrim('BCA','B') from dual;
Output	CA

7) rtrim():

Syntax	rtrim(string, [character_set])
Purpose	Returns char, with final characters removed after the last character not in the set. Set is optional, it defaults to spaces.
Example	Select rtrim('RAMA','A') from dual;
Output	RAM

8) trim():

Syntax	ltrim(leading\trailing\both[<trim_character>] FROM string)
Purpose	Removes characters from leading, trailing and both the side.
Example	Select trim(' BCA ') from dual;
Output	BCA

9) lpad():

Syntax	lpad(string, length, [character_set])
Purpose	It returns string after adding character set to the left of the string by the number specified in length.
Example	Select lpad('BCA', 10,'*') from dual;
Output	*******BCA

10) rpad():

Syntax	rpad(string, length, [character_set])
Purpose	It returns string after adding character set to the right of the string by the number specified in length.
Example	Select rpad('BCA', 10,'*') "rpad" from dual;
Output	BCA*******

11) instr():

Syntax	instr(string1,string2, [start position], [occurrence])
Purpose	It returns the occurrence of character in a given string.
Example	Select instr('information', 'n',1,1) "occurrence-1", instr('information', 'n',1,2) "occurrence-2" from dual;
Output	occurrence-1 occurrence-2 2 11

12) translate():

Syntax	translate(string1,<string_to_replace>, <replacement_string>)
Purpose	Replaces a sequence of characters in a string with another set of characters.
Example	Select translate('ORACLE','OR','12') from dual;
Output	12ACLE

(4) Date Functions:

- Oracle provides various functions to perform operations on date data type.
- You should be aware that there is a special keyword called sysdate to know the current data.
- **Ex**: select sysdate from dual;
- **Output**: 15-jun-18

1) add_months():

Syntax	add_months(date, n)
Purpose	Returns date after adding n month in date.
Example	Select add_months('10-apr-17',5) from dual;
Output	10-sep-17

2) last_day():

Syntax	last_day(date)
Purpose	Returns the last date of the month specified in function argument.
Example	Select last_day('10-apr-17') from dual;
Output	30-apr-17

3) months_between():

Syntax	months_between(date1,date2)
Purpose	Returns numbers of months between two dates date1 and date2.
Example	Select months_between('10-apr-17','10-feb-17) from dual;
Output	2

4) next_day():

Syntax	next_day(date, 'day')
Purpose	Returns the date for specified day coming after the date specified in the function argument.
Example	Select next_day('23-dec-12','sunday') from dual;
Output	30-dec-12

(5) Conversion Functions:

- Conversion functions are used to change one data type to other data type.

1) to_number():

Syntax	to_number(string)
Purpose	Converts character data type to number data type.
Example	Select to_number(substr('$100',2,3)) from dual;
Output	100

2) to_char():

Syntax	to_char(number, 'format')
Purpose	Converts number data type to character data type.
Example	Select to_char(144530,'$099,999') from dual;
Output	$144,530

3) to_char():

Syntax	to_char(date, 'format')
Purpose	Returns date in given format.
Example	Select to_char(sysdate,'dd-mm-yyyy') from dual;
Output	15-05-2018

4) to_date():

Syntax	to_date(string, 'format')
Purpose	Converts string to date value.
Example	Select to_date('14/12/2017','dd-mm-yy') from dual;
Output	14-dec-17

♦ Q-3 Explain Grouping Data in detail.

- **Group By:**
  - To create summary information or to group a data based on columns value you can use GROUP BY.
  - While grouping data you must use grouping function expression.
  - The GROUP BY clause creates a data set, containing several sets of records grouped together based on a condition.
  - The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups.
- **Features of group by clause:**
  - Group by clause comes after where clause if there is no 'where' clause it comes after from clause.
  - Group by partition the table row and create sub group within the same value in group by column.
  - Group by treats null value as separate column.
  - Group by will first sort the table on the group by column.
  - Only aggregate function can be used with group by clause. Such as sum, min, max, count, etc.
  - Column listed in group by clause need not be listed in select statement.

- **Syntax**

```
Select <column_name1>, <column_name2>,...., <Column_NameN>
Aggregate_Function (<Expression>) From <Table_Name> Where <Condition>
Group By <column_name1>, <column_name2>,...., <Column_NameN>;
```

- **Example-1**

```
Select city, count (city) from student group by city;
```

Here, above command will display city name and no. of student living in particular city.

City	Count (City)
Bhavnagar	5
Baroda	4
Surat	3

- **Example-2**

```
Select Branch_No, count (No_of_Employee) from Emp_master group by Branch_No;
```

Here, above command will display the total no of employees in each branch.

Branch_No	No_of_Employee
B1	2
B2	5
B3	4
B4	3

- **Having Clause:**
  - Having clause is used to filter a group data.
  - It is like a WHERE clause.
  - Only difference that where clause is used to filter individual row data while having clause is used to filter group data.
  - Having clause is use if and only if there is grouping clause.
- **Syntax**

```
Select <column_name1>, <column_name2>,...., <Column_NameN>
Aggregate_Function (<Expression>) From <Table_Name> Where <Condition>
Group By <column_name1>, <column_name2>,...., <Column_NameN>
Having (Condition);
```

- **Example**

```
Select city, count (city) from student group by city having count (city)>3;
```

Here, above command will display city name and no. of student living in particular city but more than 3.

City	Count (City)
Bhavnagar	5
Baroda	4

♦ Sub-query

- A sub query is a query whose results are passed as the argument for another query.
- Sub queries enable you to bind several queries together.
- In simple words, a sub query lets you tie the result set of one query to another.
- It can be used for the following purpose.
- To insert record in target table
- To create table and insert records in table created
- To update records in target table
- To create views
- To provide values for condition in where, having, in and so on used with select, update and delete statement.
- **Syntax**:
- select * from table1 where table1.somecolumn Operator (select someothercolumn from table2 where someothercolumn = somevalue)
- select * from dept order by deptno;

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

- select * from emp;

EMPNO	ENAME	JOB	SAL	DEPTNO
7369	SMITH	CLERK	800	20
7499	ALLEN	SALESMAN	1600	30
7521	WARD	SALESMAN	1250	30
7566	JONES	MANAGER	2975	20
7654	MARTIN	SALESMAN	1250	30
7698	BLAKE	MANAGER	2850	30
7782	CLARK	MANAGER	2450	10
7788	SCOTT	ANALYST	3000	20
7839	KING	PRESIDENT	5000	10
7844	TURNER	SALESMAN	1500	30
7876	ADAMS	CLERK	1100	20
7900	JAMES	CLERK	950	30
7902	FORD	ANALYST	3000	20
7934	MILLER	CLERK	1300	10

- **Simple sub-query**
- A simple sub-query is evaluated once for each table.
- You would like to select all employees whose department is located in Chicago.
- A join would be a better solution for this select, but for the purposes of illustration we will use a subquery.
- select empno, ename, job, sal, deptno from emp where deptno in (select deptno from dept where loc ='CHICAGO');

EMPNO	ENAME	JOB	SAL	DEPTNO
7900	JAMES	CLERK	950	30
7844	TURNER	SALESMAN	1500	30
7698	BLAKE	MANAGER	2850	30
7654	MARTIN	SALESMAN	1250	30
7521	WARD	SALESMAN	1250	30
7499	ALLEN S	ALESMAN	1600	30

- **Correlated Oracle sub query:**
- A correlated Oracle subquery is evaluated once **FOR EACH ROW** compare to a normal subquery which is evaluated only once for each table.
- You can reference the outer query inside the correlated subquery using an alias which makes it so easy to use.
- Let's select all employees whose salary is less than the average of all the employees' salaries in the same department.
- select ename ,sal ,deptno from emp a where a.sal < (select avg(sal) from emp b where a.deptno = b.deptno) order by deptno;

ENAME	SAL	DEPTNO
CLARK	2450	10
MILLER	1300	10
SMITH	800	20
ADAMS	1100	20
WARD	1250	30
MARTIN	1250	30
TURNER	1500	30
JAMES	950	30

- ***Using a correlated sub-query in an update:***
- Let's give these people (whose salary is less than their department's average) a raise.
- But before updating these data we want to see emp table record.
- select ename, sal, deptno from emp order by deptno, ename;

ENAME	SAL	DEPTNO
CLARK	2450	10
KING	5000	10
MILLER	1300	10
ADAMS	1100	20
FORD	3000	20
JONES	2975	20
SCOTT	3000	20
SMITH	800	20
ALLEN	1600	30
BLAKE	2850	30
JAMES	950	30
MARTIN	1250	30
TURNER	1500	30
WARD	1250	30

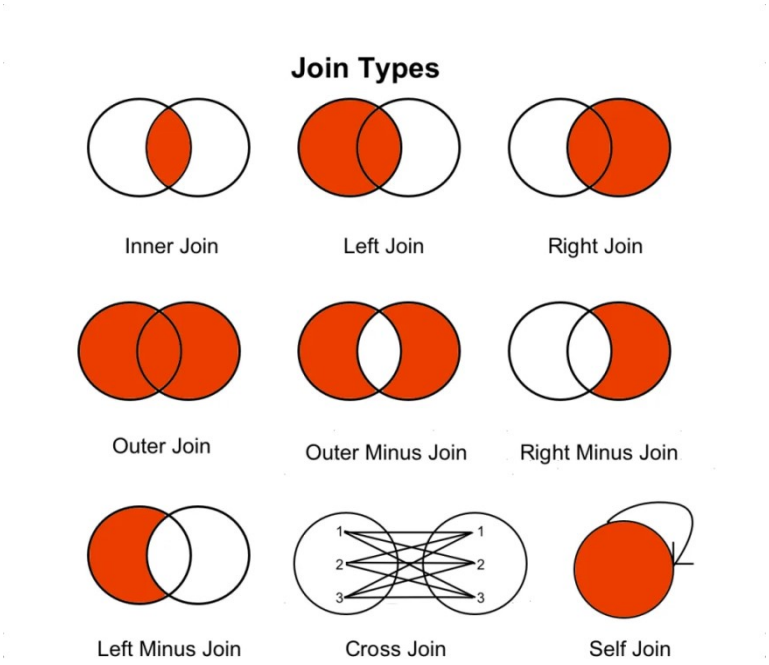
- UPDATE emp a set sal = (select avg(sal) from emp b where a.deptno = b.deptno) where sal < (select avg(sal) from emp c where a.deptno = c.deptno);
- Let see the updated record in emp table.
- select ename, sal, deptno from emp order by deptno, ename;

ENAME	SAL	DEPTNO
CLARK	2916.67	10
KING	5000	10
MILLER	2916.67	10
ADAMS	2175	20
FORD	3000	20
JONES	2975	20
SCOTT	3000	20
SMITH	2175	20
ALLEN	1600	30
BLAKE	2850	30
JAMES	1566.67	30
MARTIN	1566.67	30
TURNER	1566.67	30
WARD	1566.67	30

- ***Using a correlated subquery in a delete***
- Delete from emp a where a.sal = (select max(sal) from emp b where a.deptno = b.deptno);

♦ Join and types of join

- One of the most powerful features of SQL is its capacity to retrieve information from multiple tables.
- Without this feature one need to store unnecessary data into multiple tables which in turn create problem of inconsistency in database.
- The join statement of SQL enables you to create smaller, more specific table that are easier to maintain then larger tables.
- A join is a query that combines rows from two or more tables, views.
- Oracle Database performs a join whenever multiple tables appear in the FROM clause of the query.



- **1. *Equi join***
- Equi join is used to retrieve data from two or more then two tables based on the equality of value of the same field available in both table.
- Equi join returns only that row from both table which satisfy given condition in where clause.
- An **equijoin** is a join with a join condition containing an equality operator (=).
- SELECT * FROM employee, department where employee.DepartmentID=department.DepartmentID

- **2. *Cross joins***
- This join represent Cartesian product between two tables appeared in query.
- This type of join does not require common field in both tables appeared in query.
- Cross join query does not contain any condition where clause.
- SELECT * FROM employee, department;
- Above query return total rows_in_first_table X total rows_in_second_table.
- It returns each record in first table with each record in second table.

- **2.1** ***Self Join (join to itself):***
- Self join is used to execute complex queries that can be returned using single SQL statement.
- Self join is used to join table with itself.
- In this join same table appears twice in the FROM clause.
- This query is executed by making two copies of same table in memory which then can be compared to produce final result.
- Suppose you need to find employee name with its manager name. Both fields are in same table.

- **3.** ***Inner join***
- An **inner join** (sometimes called a **simple join**) is a join of two or more tables that returns only those rows that satisfy the join condition. It is same as equi joins.
- Example:
- SELECT * FROM employee, department WHERE employee.DepartmentID = department.DepartmentID

- **4.** ***Outer join***
- An outer join does not require each record in the two joined tables to have a matching record.
- The joined table retains each record—even if no other matching record exists.
- Outer joins subdivide further into left outer joins, right outer joins, and full outer joins, depending on which table(s) one retains the rows from (left, right, or both)

- **4.** ***(A)Left outer join***
- The result of a left outer join (or simply left join) for table A and B always contains all records of the "left" table (A), even if the join-condition does not find any matching record in the "right" table (B).
- This means that if the ON clause matches 0 (zero) records in B, the join will still return a row in the result but with NULL in each column from B.
- This means that a left outer join returns all the values from the left table, plus matched values from the right table (or NULL in case of no matching join predicate).
- If the left table returns one row and the right table returns more than one matching row for it, the values in the left table will be repeated for each distinct row on the right table.
- SELECT * FROM employee, department where employee.DepartmentID = department.DepartmentID

- **4.** ***(B)Right outer joins:***
- A right outer join (or right join) closely resembles a left outer join, except with the treatment of the tables reversed.
- Every row from the "right" table (B) will appear in the joined table at least once. If no matching row from the "left" table (A) exists, NULL will appear in columns from A for those records that have no match in B.
- A right outer join returns all the values from the right table and matched values from the left table (NULL in case of no matching join predicate).

SELECT * FROM employee department where employee.DepartmentID(+) = department.DepartmentID

- **4. *(C)Full outer join:***
- A full outer join combines the results of both left and right outer joins.
- The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

♦ Views

- Views are database objects that look like tables but views are created from a SELECT statement run on one or more tables.
- In other words, a view is a subset of data from one or more tables.
- Tables used in view are known as base table.
- A view does not contain its own data; the contents of a view are dynamically retrieved from the tables on which it is based.
- View does not occupy physical space in database.
- A view is sometimes referred to as a stored query.
- Views can increase the usability of the database by making complex queries into simple query.
- For example, when user needs to do some operations on group of related tables, he can create views that combine records from group of tables and perform operation on it very easily.
- Views can also be used to restrict access to certain rows or columns of a table.
- For example, the DBA can create a view against the EMPLOYEES table that excludes the SALARY column and can make this view available to departments that need to see employee information but should not see salary information.

- To create view one need to use following syntax.

```
CREATE [OR REPLACE] VIEW <view name>
   [<Column list> ] AS
   <table expression> [where <condition>]
```

- For example we want a view that shows all columns and all record from emp table.
- **Create view allempview as select * from emp**

- For example we want to create view that shows employee no,employee name and salary from all record of employee table.
- **Create view empview as select EMPNO , ENAME, SAL from emp**

- For example we want to create view that shows employee no, name of those employee whose salary is less than 10000 but greater than 5000.
- **Create view empview2 as select EMPNO, ENAME from emp where SAL>5000 and SAL<10000**

- Now we need to modify above view to include record of those employee whose salary is less than 20000.
- **Create or replace empview2 as select EMPNO, ENAME from emp where SAL<20000.**
- We can use view to perform following operation on table or group of table.
- Comments, Delete, Insert, Lock table, Update, select

- Let us use empview that we have just created to retrieve information from emp table. We want to show employee no, employee name whose salary is less then 10000

- **select empno,ename from emp where salary<10000**

- We can also apply condition on records that we want to retrieve from view.
- Now let us use update command using same view

- **Update empview set ename='rajesh where ename='raj'**

- Above command update emp table using its view empview and change ename to rajesh for the record in which ename is raj
- Let's try delete command using same view

- **Delete from empview where ename='rajesh'**

- Above command delete record from emp table using its view empview for above given condition.
- Let's try insert command using allempview

- **Insert into allempview values(1077,1011,'nishant','sales',1000,'12-jan-07',12500,1)**

- Above command insert new row in emp table using allempview.

- **DROP VIEW command**
- The DROP VIEW simply drop the VIEW from the database. It does not affect base table(s) used in view at all. To drop the view use following syntax.

- **DROP VIEW viewname**

- Now delete all views that we created previously using above syntax
- Drop view allempview
- Drop view empview
- Drop view empview2

♦ INDEX

- Once we create tables in any real life database, we start storing millions of record in it. All these record are used to retrieve information in it using select statement or it may be changes using update statement.
- If index is not available, oracle search for given record using sequential search method on table data. This is very slow method of finding record.
- Now question is that how we can do all these operation at better speed and in less time?. We can do this using INDEX.
- Index is an ordered list of the contents of column (or group of column) of a table.
- Index is based on data in tables.
- An index is optional structure associated to tables that increase the data recovery performance.

- Indexes can be created on single field or group of field.
- Once index is created it is automatically used and updated by oracle.
- Index has two dimensional matrix which is independent from table on which index is created.
- This matrix has two field. One field store the value of index field into ascending order and second field store location of record in the oracle database.
- The record in the index are sorted in the ascending order of index column so oracle quickly locate particular record from table if select query has been given with where clause on field which has index.
- Adding indexes to your database enables SQL to use the Direct Access Method. SQL use a tree-like structure to store and retrieve the index's data.

- **Index are used for the following reason**
- To enforce referential integrity constraint by using unique or primary key keywords.
- To provide reordering of data based on the content of index's field or fields
- To increase execution speed of query.

- To create index on field(s) use following syntax.
- **Create index [unique] <indexname> on <table> (column1[ASC|DESC],[column2][column…])**

- Before we create index on emp field first issue following command
- **Select * from emp**
- Above statement show all record in natural order of insertion because there is no index table or order by clause in above statement.

- Now to create index on ename field of emp table, the create index statement would be like this.
- **CREATE INDEX ename_index on emp(ename);**

- **DROP INDEX command**
- Drop index commands drop index from database. It syntax is following

- Syntax
- **DROP INDEX <index name>**

- Example
- To drop ename_index, one should give following commands.
- **Drop index ename_index.**

- You cannot create index that uses only one column if it was already used by another index.
- However you can create composite index that include columns that was already used by another index.
- Composite index include two column in creation of index however only one physical index is created.

- For example we want to create composite index on empno and ename column of emp table
- **Create index empno_ename_index on emp(empno,ename)**
- Now use following SQL command

- **Select * from emp;**
- This command arranges all record in ascending order of empno and then on ename.

♦ SEQUENCES:

- A sequence generates a sequential list of unique numbers for table's columns.
- A sequence generates a unique key for the primary key of a table.
- It is an automatic counter that is used to give unique value when we perform insert operation to add new record in table.
- It also generate correct unique value when multiple users insert record in table same time.
- User can generate unique number up to 38 digits using sequences.
- Sequence's definition is stored in a table of database.
- Sequences can begin and end with any value, can be ascending or descending, and can skip (increment) a specified number between each value in the sequence.

- The basic syntax for CREATE SEQUENCE is as follows:
- **CREATE SEQUENCE <sequencename> [Start with <integer_value>] [Increment by <integer_value>] [MINVALUE <integer_value> | NOMINVALUE] [MAXVALUE <integer_value> NOMAXVALUE] [Cycle|nocycle]**

- If all optional parameters are omitted, then sequence starts with one and increases by increments of one, with no upper boundary.

- For example we want sequence for empno field in emp table.
- **Create sequence empno_sequence start with 1 increment by 2 maxvalue 1000 nocyle**

- Above sql statement create empno_sequence with given specification.
- Now use above sequence to insert record in emp table
- **Insert into emp values(empno_sequence.NEXTVAL, 1011,'nita,'production',1000,'18-jan-07',12500,1)**

- Sequence.NEXTVAL is used to increase value of sequence and then it retrieve that value for in insert command.
- We can find current value of sequence using sequencename.CURRVAL in select statement.
- **Select empno_sequence .CURRVAL from dual.**

- **Checking user list of sequence**
- To check sequence that we were created we can use following command.
- **Select * from user_sequence**

- **Changing sequence**
- With alter sequence command user can changes some of the parameters in sequence.
- **ALTER SEQUENCE sequence_name [INCREMENT BY int] [MAXVALUE int | NOMAXVALUE][MINVALUE int | NOMINVALUE] [CYCLE | NOCYCLE]**
- For example we want our empno_sequence to be incremented by 1 and it maximum value should be 12500.
- **Alter sequence empno_sequence increment by 1 maxvalue 12500**
- **Dropping sequence**

- To drop sequence we can use following syntax

- **Drop sequence <sequence_name>**

- Now delete empno_sequence that was created previously
- **Drop sequence empno_sequence**

♦ Synonyms

- A synonym is an alias for a schema object. Synonyms can provide a level of security by masking the name and owner of an object and by providing location transparency for remote objects of a distributed database. Also, they are convenient to use and reduce the complexity of SQL statements for database users.
- **Create Synonym**
- Use the CREATE SYNONYM statement to create a synonym, which is an alternative name for a table, view, sequence, procedure, stored function, package, materialized view, Java class schema object, user-defined object type, or another synonym.

- Syntax
- CREATE [OR REPLACE] [PUBLIC] SYNONYM [schema1.]synonym FOR [schema2.]object

- Example
- CREATE SYNONYM SYEMP FOR EMP;

- **View Synonym**
- To view synonym

- Syntax
- SELECT * FROM SYNONYMS;

- Example
- SELECT * FROM SYEMP;
- SELECT COUNT(*) FROM CUSTOMERS;
- SELECT EMPNO, EMPNAME FROM SYEMP;

- **Drop Synonym**
- The DROP SYNONYM statement removes a synonym from the database.

- Syntax
- DROP SYNONYM SYNONYMNAME;